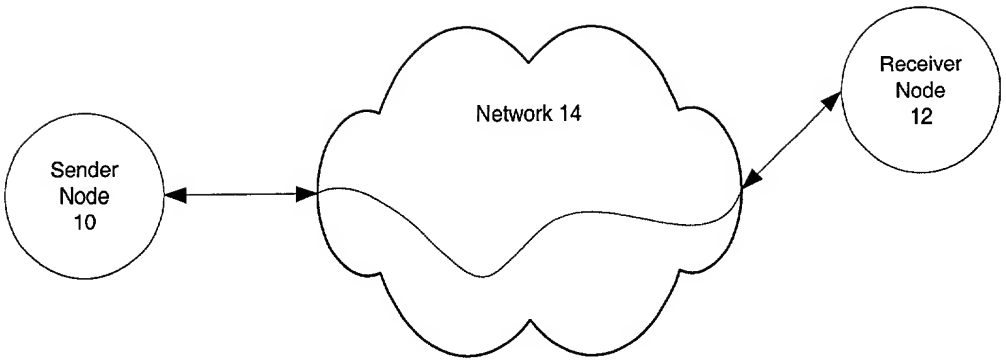
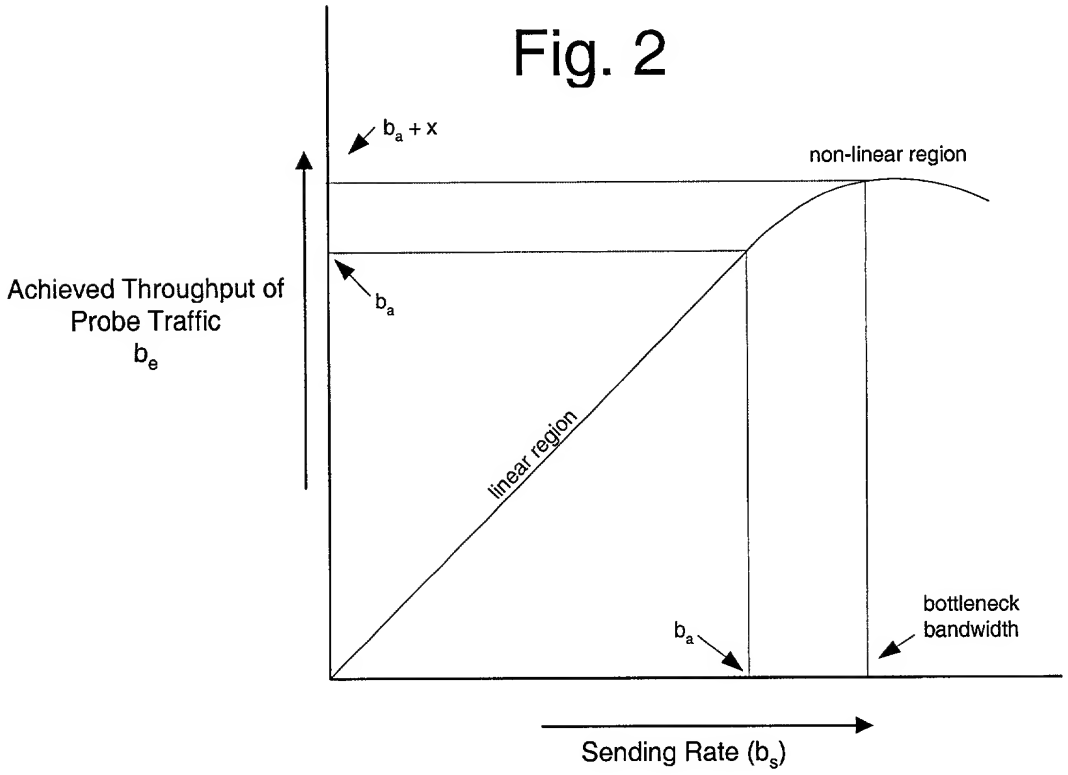

Sender
Node
10
Network 14
Receiver
Node
12

Fig. 1

Fig. 2


$b_a + x$
non-linear region
Achieved Throughput of
Probe Traffic
b_e
b_a
linear region
b_a
bottleneck
bandwidth
Sending Rate (b_s)

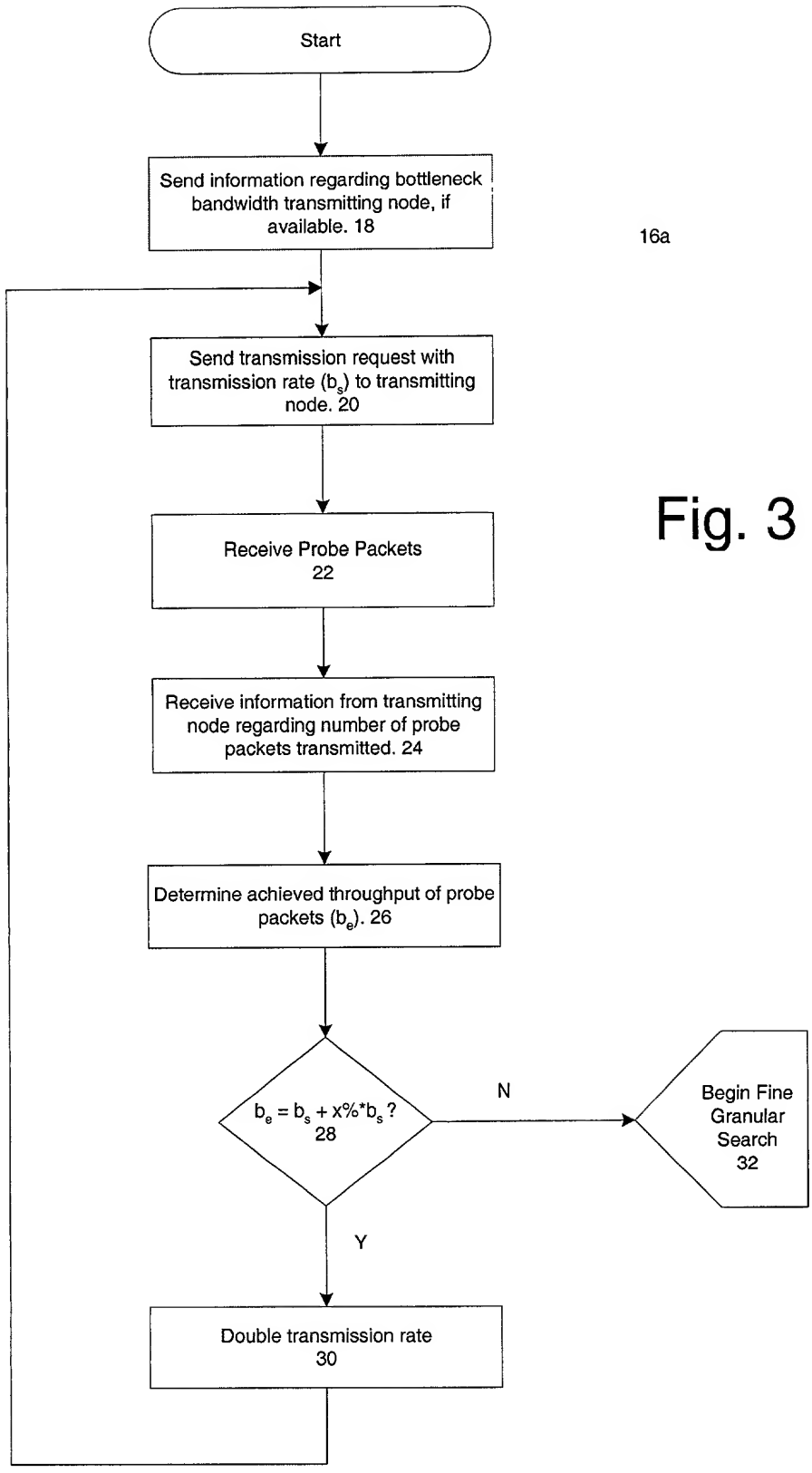
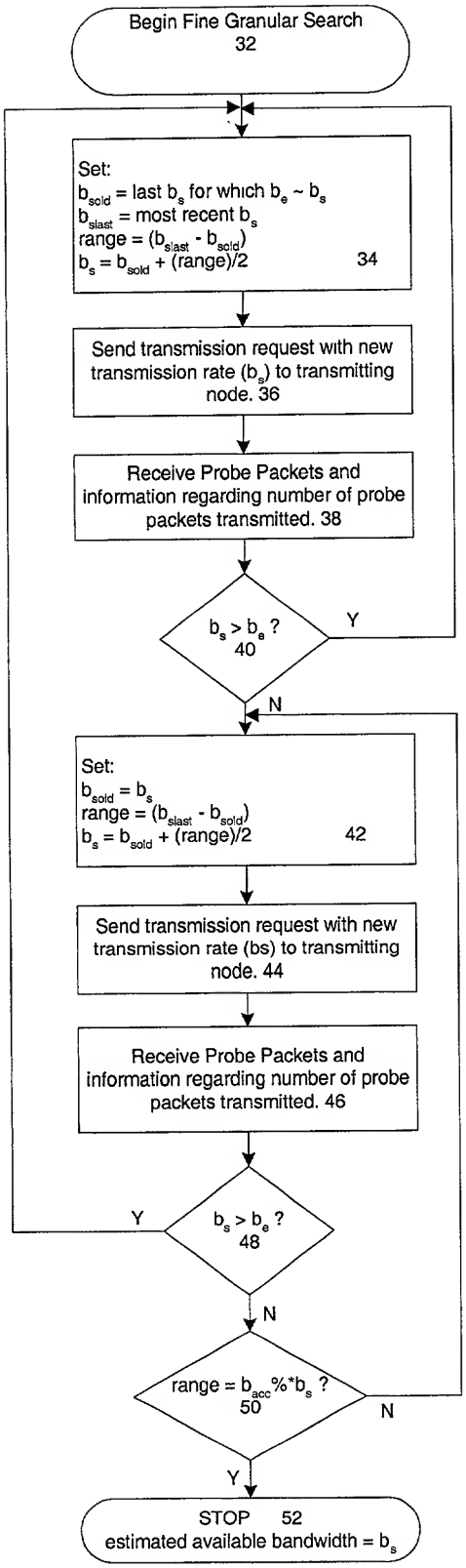

Start
Send information regarding bottleneck bandwidth transmitting node, if available. 18
16a
Send transmission request with transmission rate (b_s) to transmitting node. 20
Receive Probe Packets 22
Receive information from transmitting node regarding number of probe packets transmitted. 24
Determine achieved throughput of probe packets (b_e). 26
$b_e = b_s + x\%*b_s$? 28
N
Begin Fine Granular Search 32
Y
Double transmission rate 30

Fig. 3


Begin Fine Granular Search
32
Set:
b_{sold} = last b_s for which $b_e \sim b_s$
b_{slast} = most recent b_s
range = (b_{slast} - b_{sold})
b_s = b_{sold} + (range)/2
34
Send transmission request with new transmission rate (b_s) to transmitting node. 36
Receive Probe Packets and information regarding number of probe packets transmitted. 38
$b_s > b_e$?
40
Y
N
Set:
b_{sold} = b_s
range = (b_{slast} - b_{sold})
b_s = b_{sold} + (range)/2
42
Send transmission request with new transmission rate (bs) to transmitting node. 44
Receive Probe Packets and information regarding number of probe packets transmitted. 46
$b_s > b_e$?
48
Y
N
range = b_{acc}%*b_s ?
50
N
Y
STOP 52
estimated available bandwidth = b_s

16b

Fig. 4

Fig. 5

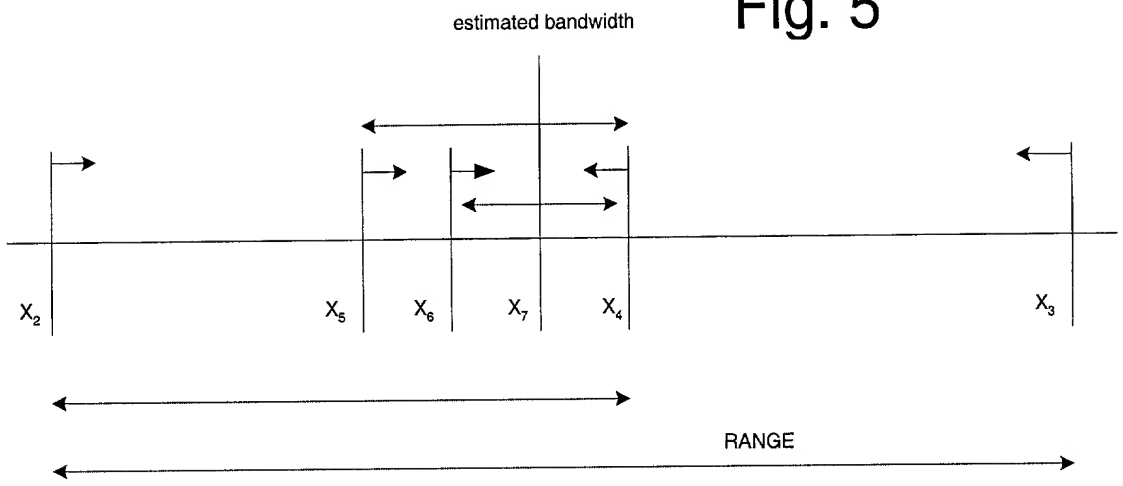
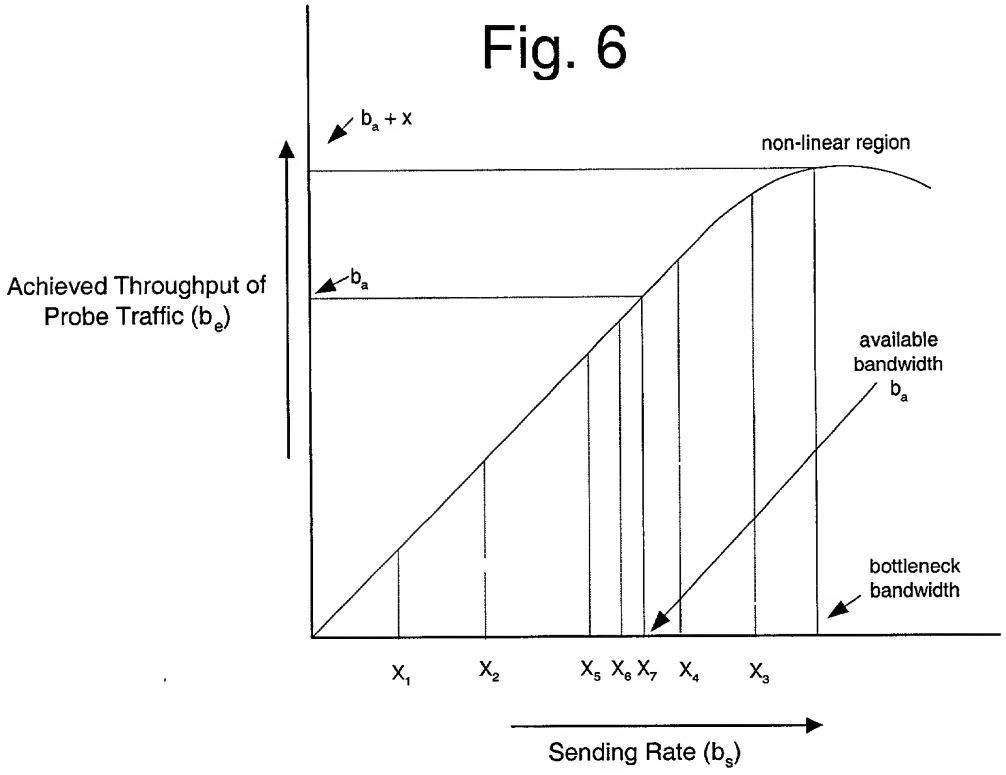

estimated bandwidth
X2
X5
X6
X7
X4
X3
RANGE

Fig. 6


ba + x
non-linear region
ba
Achieved Throughput of Probe Traffic (be)
available bandwidth ba
bottleneck bandwidth
X1
X2
X5
X6
X7
X4
X3
Sending Rate (bs)

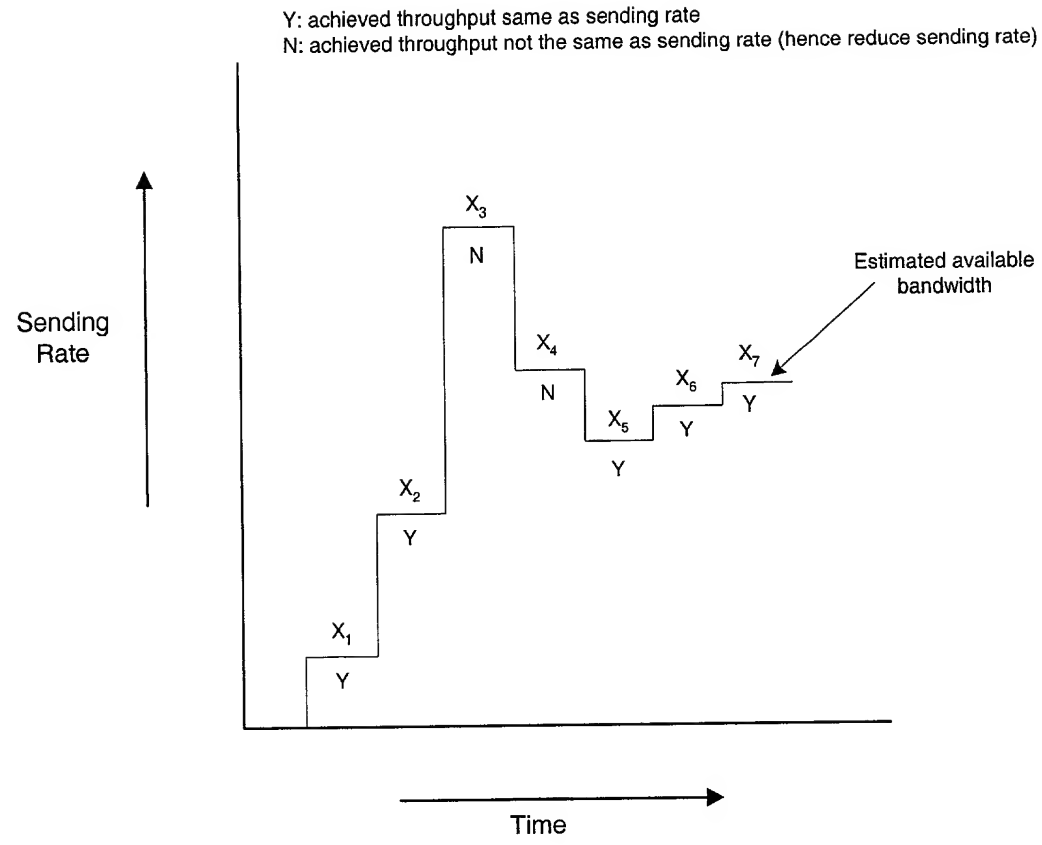


Fig. 7

```
01 Procedure ReceiverInitialize(sender)
02 open up TCP socket with the sender
03 open up UDP socket
04 rnd_no ← 0
```

Fig. 8

Procedure exp_growth()

global constants : fraction_start, increase_factor, fixed_minimum_rate

```
01 when Sender initiates connection
02     receiver acks and asks sender for bottleneck bandwidth
03     wait for bottleneck bandwidth information from sender
04     receives bottleneck bandwidth information
05 if (bottleneck bandwidth available)
06     start_sending_rate = fraction_start*bottleneck bandwidth
07     increase_factor = 2
08 else
09     start_sending_rate = fixed_minimum_rate
10     increase_factor = 10
11 control_info.sending_rate = start_sending_rate
12 control_info.time = random(min_duration, max_duration)
13 control_info.rnd = ++rnd
13 send control_info to the sender
14 listen to probe packets on UDP socket and control packets on TCP socket
15 if (received probe_packet)
16     if (probe_packet is first packet of rnd)
17         t_first ← time when received
18         seq_first ← sequence number of packet
16     if (probe_packet is last packet of rnd)
17         t_last ← time when received
18         seq_last ← sequence number of packet
19     if (packet.seq_no < sequence number of last packet of same rnd)
19         packet_reordering ← TRUE
20     if(received control_packet)
21         total_sent_bytes = control_packet.total_bytes
22     if (seq_last < pktID AND total_sent_bytes > total_recvd_bytes)
23         packet_loss ← TRUE
24 if(packet_loss || packet_reordering)
25     try one more round if not already tried for 3 rounds with same sending rate
26     control_info.sending_rate = last_sending_rate
27     control_info.rnd = ++rnd
28     control_info.time = random(min_duration, max_duration)
29     go to 13
```

cont'd next page

Fig. 9a

Procedure exp_growth() contd...

```
30 else
31     achieved throughput = total_sent_bytes/(t_last - t_first)
32     if((achieved throughput - sent rate)/sent rate) * 100 < ε
33         control_info.sending_rate = increase_factor*last_sending_rate
34         control_info.cnd = ++cnd
35         control_info.time = random(min_duration, max_duration)
36         min_range = control_info.sending_rate
37         if(control_info.sending_rate > bottleneck bandwidth)
38             max_range ← bottleneck bandwidth
39         else
40             go to 13
41             return min_range, max_range
42     else
43         max_range ← control_info.sending_rate
44         return min_range, max_range
```

Fig. 9b

```
Procedure fine_granular_search()
global constants : fraction_start, increase_factor, fixed_minimum_rate
                   min_duration, max_duration, acceptable_fraction
                   no_bw_acceptable_fraction

01 range ← max_range − min_range
02 control_info.sending_rate ← min_range + range/2
03 control_info.time = random(min_duration, max_duration)
04 control_info.rnd = ++rnd
05 send control_info to the sender
06 listen to probe packets on UDP socket and control packets on TCP socket
07 if (received probe_packet)
08     if (probe_packet is first packet of rnd)
09         t_first ← time when received
10         seq_first ← sequence number of packet
11     if (probe_packet is last packet of rnd)
12         t_last ← time when received
13         seq_last ← sequence number of packet
14     if (packet.seq_no < sequence number of last packet of same rnd)
15         packet_reordering ← TRUE
16     if(received control_packet)
17         total_sent_bytes = control_packet.total_bytes
18     if (seq_last < pktID AND total_sent_bytes > total_recvd_bytes)
19         packet_loss ← TRUE
20 if(packet_loss || packet_reordering)
21     try one more round if not already tried for 3 rounds with same sending rate
22     control_info.sending_rate = last_sending_rate
24     control_info.rnd = ++rnd
25     control_info.time = random(min_duration, max_duration)
26     go to 13
27 else
28     achieved throughput = total_sent_bytes/(t_last − t_first)
29     if((achieved throughput - sent rate)/sent rate) * 100 < ε
30         min_range ← current_sending_rate
cont'd next page
```

Fig. 10a

Procedure fine_granular_search() contd....

```
31      control_info.sending_rate = min_range + range/2
32      range = max_range - min_range
33      if (bottleneck bandwidth information available)
34          if (range/bottleneck_bandwidth < acceptable_fraction)
35              return min_range + range/2
36          else
37              control_info.time = random(min_duration, max_duration)
38              control_info.rnd = ++rnd
39              go to Line 5
40      else
41          if (range/min_range_bandwidth < no_bw_acceptable_fraction)
42              return min_range + range/2
43          else
44              control_info.time = random(min_duration, max_duration)
45              control_info.rnd = ++rnd
46              go to Line 5
47  else
48      max_range ← current_sending_rate
49      range = max_range - min_range
50      control_info.sending_rate = min_range + range/2
```

Fig. 10b

```
Procedure sender(receiver)
global constants : fraction_start, increase_factor, fixed_minimum_rate
                   min_duration, max_duration, acceptable_fraction
                   no_bw_acceptable_fraction, pkt_size

01 Initiate tcp connection with the receiver
02 After Receiver acks
03     open UDP socket for sending probe packets
04     sends receiver information about bottleneck bandwidth if available
05     when control_info is received over the tcp channel, terminates if
       received TERMINATION signal, prints the available bandwidth information
       provided by the sender
06     sending_rate = control_info.sending_rate
07     tot_time = control_info.time
08     rnd_no = control_info.rnd
09 avg_pkt_rate = sending_rate/packet_size
10 pkt_id = 0
10 starting_time = PRESENT_TIME
11 prepare a pkt
12 pkt.pkt_id = pkt_id
13 pkt.rnd = rnd_no
14 pkt.size = pkt.size
15 send the packet
16 if (PRESENT_TIME - starting_time) < tot_time
17     wait( 1/avg_pkt_rate)
18 else
19     control_info.sending_rate = sending_rate
20     control_info.time = PRESENT_TIME - starting_time
21     control_info.rnd = rnd_no
22     control_info.last_id = pkt_id
```

Fig. 11